Tarea 4

Redes Sociales y Económicas

Sergi Fornés, Jordi Vanrell & Daniel Ramos

1) Generate an undirected random graph using the model of "preferential attachment" (`sample_pa( )`) of 1000 nodes. With $\beta = 0.1$, $\gamma = 0.1$, generate a SIR pandemic (iterative method). The initial infected nodes should be the 10 highest using the `eigen_centrality( )`. Compare the results to when the initial nodes are at random. Reduce or increase β and compare.

```
# Generamos el grafo
set.seed(2021)
gf <- sample_pa(1000, directed=F)
# Obtenemos la matriz de adyacencia
A <- as_adjacency_matrix(gf)
# Especificamos los parámetros que usaremos en el modelo SIR
beta <- 0.1
beta_b <- 0.05
beta_B <- 0.4
gamma <- 0.1

# Creamos los vectores que contienen el estado de los nodos
I <- rep(1, 1000)
# Buscamos los 10 nodos más centrales
positions <- order(eigen_centrality(gf)$vector, decreasing = TRUE)[1:10]
x0 <- rep(0, 1000)
x0[positions] <- 1
s0 <- I - x0
r0 <- rep(0, 1000)
v <- list(s = s0, x = x0, r = r0)
v_b <- list(s = s0, x = x0, r = r0)
v_B <- list(s = s0, x = x0, r = r0)

# Creamos los vectores que guardarán la cantidad de nodos en función de su estado
count <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))
count_b <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))
count_B <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))

# Creamos la función que actualiza el estado de los nodos
upd <- function(s, x, r, b, g) {
  probs_x <- pmin(pmax(b * s * (A %*% x) - g * x, 0), 1)
  probs_r <- pmin(pmax(g * x, 0), 1)
  for(i in seq(length(x))) {
    if(x[i] == 1 & rbinom(1,1,probs_r[i]) == 1) {
      r[i] <- 1
      x[i] <- 0
    }
```

```
    if(s[i] == 1 & rbinom(1,1,probs_x[i]) == 1) {
      x[i] <- 1
      s[i] <- 0
    }
  }
  return(list(s = s, x = x, r = r))
}

# Guardamos la cantidad de nodos en función de su estado durante n periodos
n <- 65
for(i in 2:n) {
  v <- upd(v$s, v$x, v$r, beta, gamma)
  count$s[i] <- sum(v$s)
  count$x[i] <- sum(v$x)
  count$r[i] <- sum(v$r)
  v_b <- upd(v_b$s, v_b$x, v_b$r, beta_b, gamma)
  count_b$s[i] <- sum(v_b$s)
  count_b$x[i] <- sum(v_b$x)
  count_b$r[i] <- sum(v_b$r)
  v_B <- upd(v_B$s, v_B$x, v_B$r, beta_B, gamma)
  count_B$s[i] <- sum(v_B$s)
  count_B$x[i] <- sum(v_B$x)
  count_B$r[i] <- sum(v_B$r)
}

count <- data.frame(count$s, count$x, count$r,
                    count_b$s, count_b$x, count_b$r,
                    count_B$s, count_B$x, count_B$r)

# Creamos el gráfico
plot <- ggplot(count) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.s), color = "blue", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.x), color = "red", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.r), color = "green", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.s), color = "blue", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.x), color = "red", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.r), color = "green", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count.s), color = "blue") +
  geom_point(aes(x = seq(nrow(count)), y = count.x), color = "red") +
  geom_point(aes(x = seq(nrow(count)), y = count.r), color = "green") +
  ggtitle("Nodos iniciales más centrales") +
  xlab("t") +
  ylab("Cantidad de nodos")


# Hacemos lo mismo pero con los 10 primeros nodos elegidos al azar
positions <- sample(1:1000, 10)
x0 <- rep(0, 1000)
x0[positions] <- 1
s0 <- I - x0

v <- list(s = s0, x = x0, r = r0)
v_b <- list(s = s0, x = x0, r = r0)
```

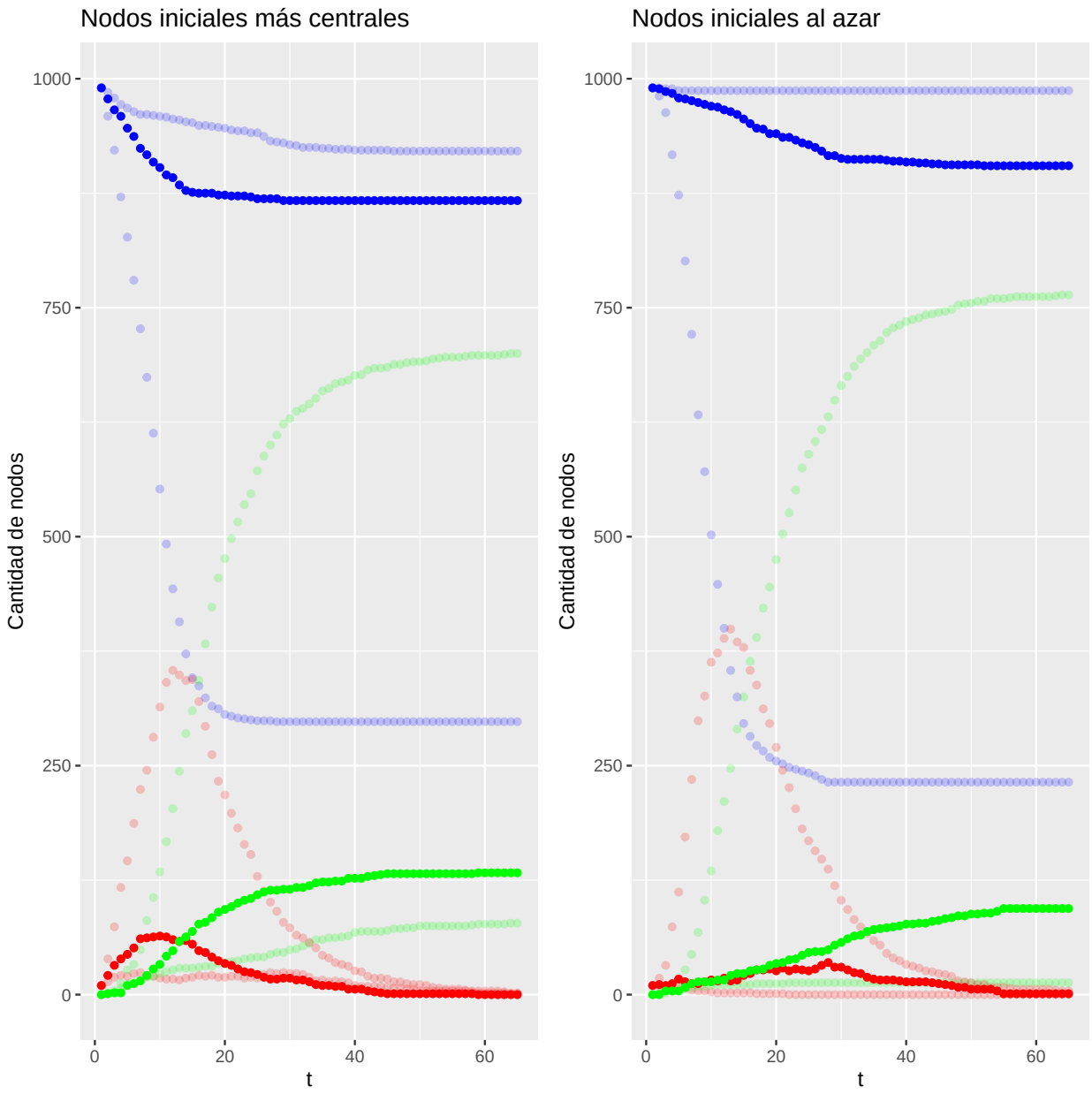
```r
v_B <- list(s = s0, x = x0, r = r0)
count <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))
count_b <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))
count_B <- list(s = sum(v$s), x = sum(v$x), r = sum(v$r))

n <- 65
for(i in 2:n) {
  v <- upd(v$s, v$x, v$r, beta, gamma)
  count$s[i] <- sum(v$s)
  count$x[i] <- sum(v$x)
  count$r[i] <- sum(v$r)
  v_b <- upd(v_b$s, v_b$x, v_b$r, beta_b, gamma)
  count_b$s[i] <- sum(v_b$s)
  count_b$x[i] <- sum(v_b$x)
  count_b$r[i] <- sum(v_b$r)
  v_B <- upd(v_B$s, v_B$x, v_B$r, beta_B, gamma)
  count_B$s[i] <- sum(v_B$s)
  count_B$x[i] <- sum(v_B$x)
  count_B$r[i] <- sum(v_B$r)
}

count <- data.frame(count$s, count$x, count$r,
                    count_b$s, count_b$x, count_b$r,
                    count_B$s, count_B$x, count_B$r)

plot2 <- ggplot(count) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.s), color = "blue", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.x), color = "red", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_b.r), color = "green", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.s), color = "blue", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.x), color = "red", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count_B.r), color = "green", alpha = 0.2) +
  geom_point(aes(x = seq(nrow(count)), y = count.s), color = "blue") +
  geom_point(aes(x = seq(nrow(count)), y = count.x), color = "red") +
  geom_point(aes(x = seq(nrow(count)), y = count.r), color = "green") +
  ggtitle("Nodos iniciales al azar") +
  xlab("t") +
  ylab("Cantidad de nodos")


# Imprimimos los gráficos
grid.arrange(plot, plot2, nrow = 1)
```



Por una parte, los puntos de color azul representan los nodos susceptibles, los de color rojo representan los infectados, y los de color verde representan los recuperados. Por otra, los puntos opacos son del modelo SIR con $\beta = 0.1$ y $\gamma = 0.1$, mientras que los puntos transparentes se han obtenido con valores de $\beta = 0.05$ y $\beta = 0.4$.

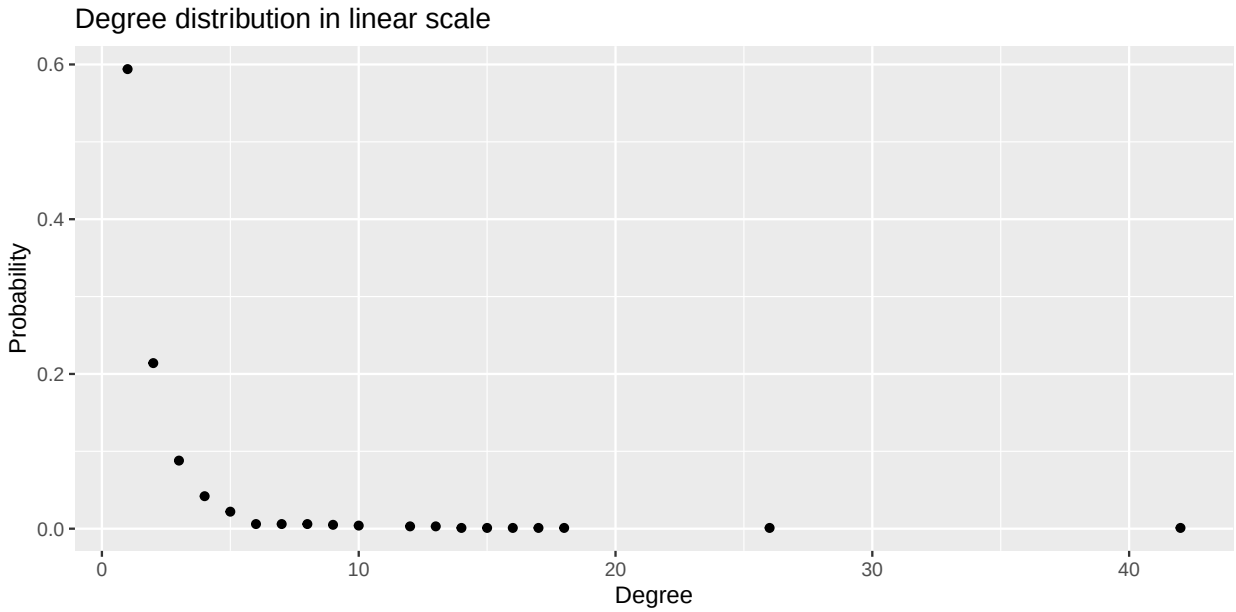
En los gráficos se puede ver la importancia de que los nodos iniciales se encuentren en el centro del grafo. En ese caso, y con valores β pequeños, la infección se produce más rápidamente y llega a más nodos. Pero si elegimos una tasa de contagio β suficientemente alta, parece que la cantidad total de nodos infectados es mayor si los nodos infectados inicialmente se encuentran repartidos al azar en el grafo.

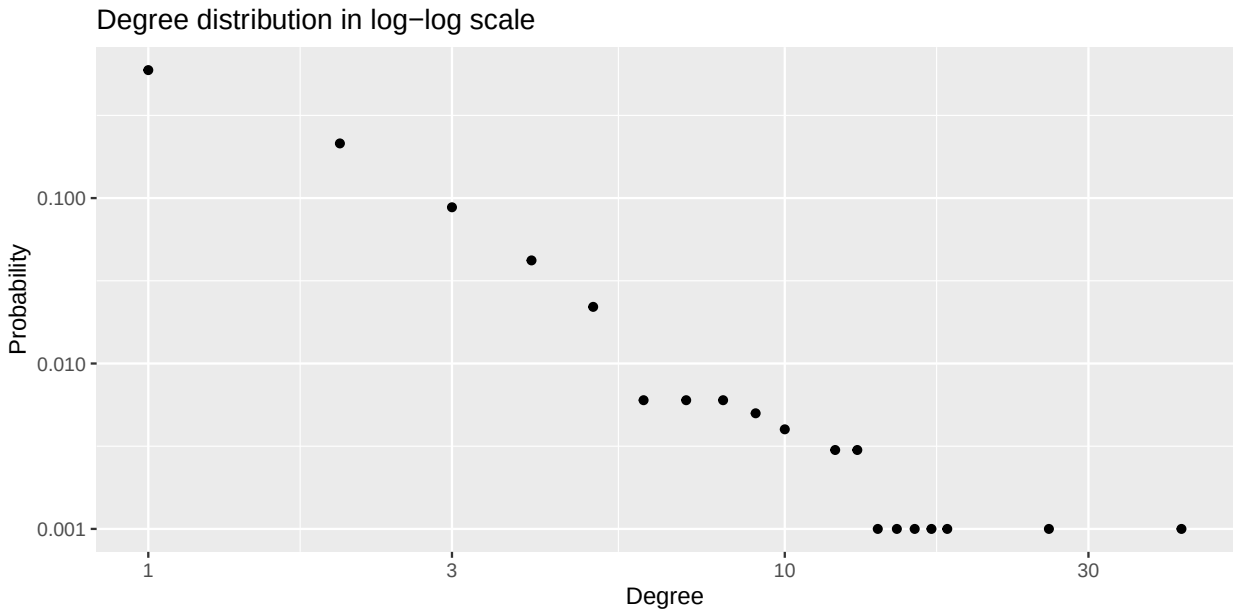
---

2) Consider the random graph generated in the previous exercise.

a) Plot its degrees distribution in linear and in log-log scale. Which is more helpful to understand this distribution?

```
dist <- degree_distribution(gf)
deg_dist <- data_frame(deg = seq(0,length(dist)-1), prob = dist) %>%
  filter(prob > 0)
plot <- ggplot(deg_dist) +
  geom_point(aes(x = deg, y = prob)) +
  ggtitle("Degree distribution in linear scale") +
  xlab("Degree") +
  ylab("Probability")
plot2 <- ggplot(deg_dist) +
  geom_point(aes(x = deg, y = prob)) +
  scale_x_log10() +
  scale_y_log10() +
  ggtitle("Degree distribution in log-log scale") +
  xlab("Degree") +
  ylab("Probability")
grid.arrange(plot, plot2, nrow = 2)
```





El gráfico en escala log-log es más fácil de interpretar que en escala lineal.

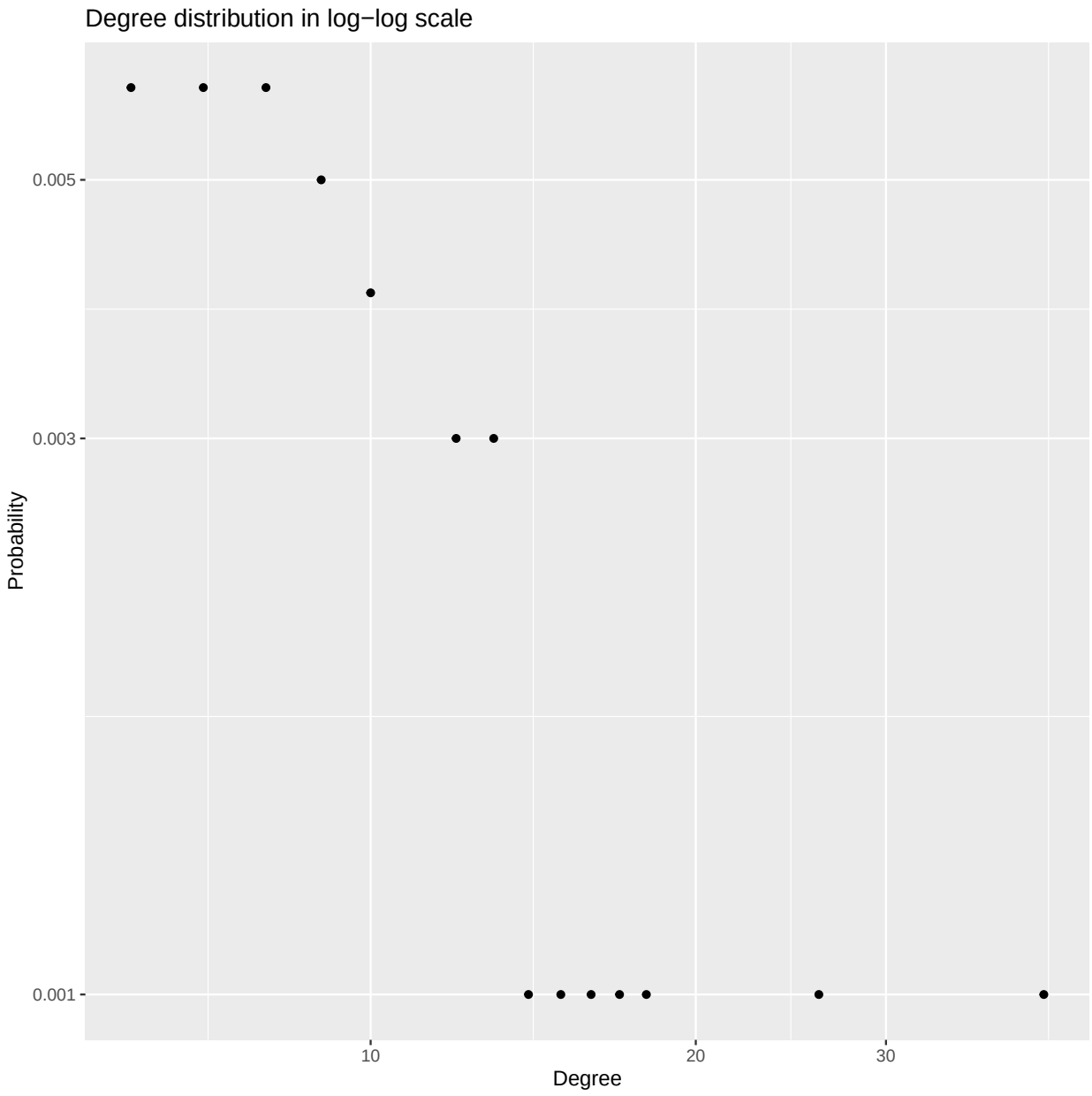
---

b) Does the degree distribution follows a Power Law? And if we consider only the nodes with degree above 5? (or 10? or 100?)

En el gráfico log-log del apartado *(a)* parece que los grados siguen una recta, por lo que es un indicio que la distribución puede seguir una ley potencial. Veamos lo que pasa cuando consideramos unicamente los grados mayores de 5.

```
ggplot(filter(deg_dist, deg > 5)) +
  geom_point(aes(x = deg, y = prob)) +
```

```
  scale_x_log10() +
  scale_y_log10() +
  ggtitle("Degree distribution in log-log scale") +
  xlab("Degree") +
  ylab("Probability")
```

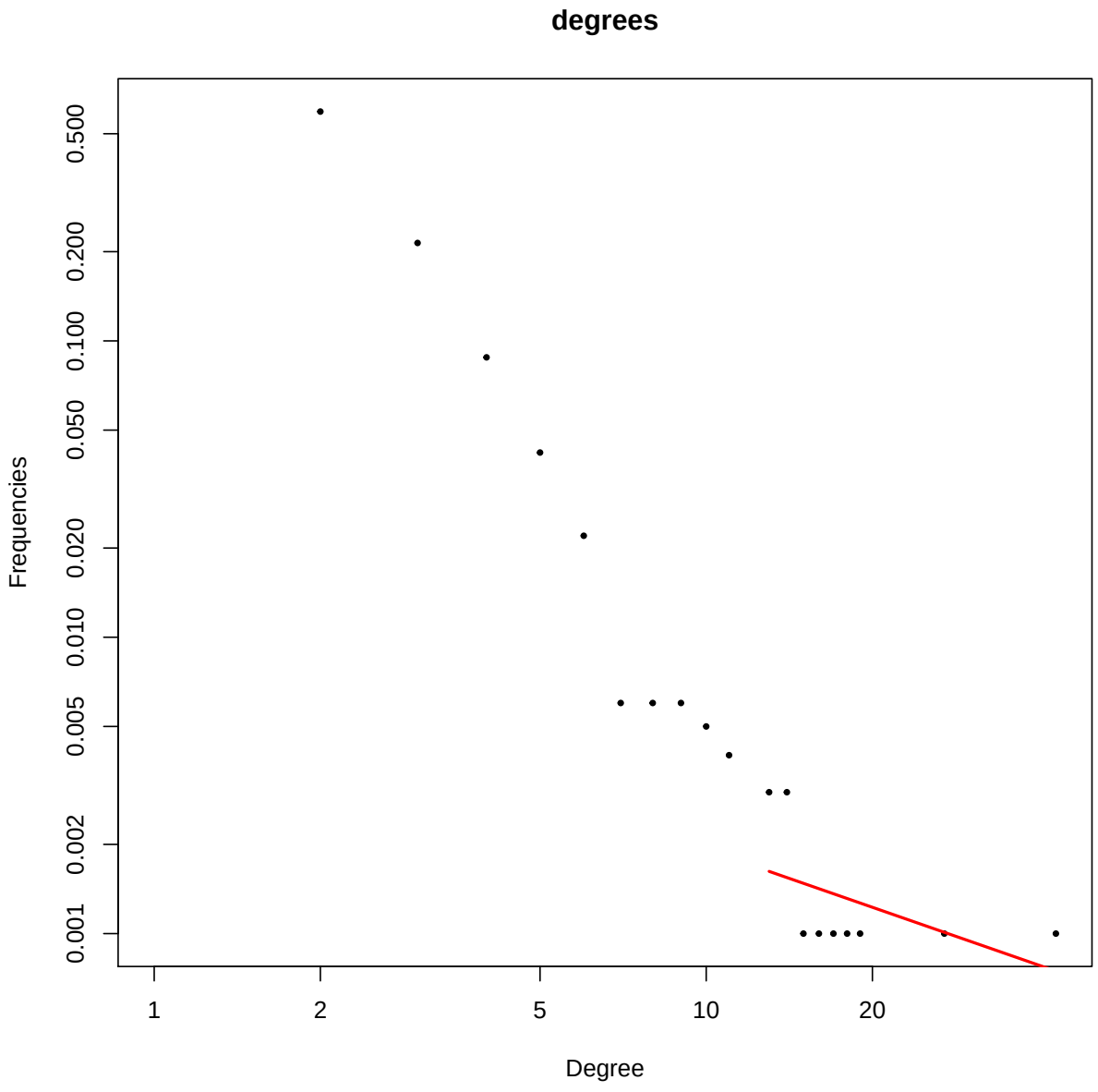


En este caso no queda tan claro que los grados siguen una recta. Hay muchos valores que son 0 o 0.001 debido a que hay muy pocos nodos de grado mayor de 13, pero si el grafo fuera mayor a lo mejor se seguiría viendo la relación lineal.

---

c) Find the best line that approximates the degree distribution after degree 10 (or 5?) using linear regression (`lm()`) on the log-log plane. Don't worry, it is almost all done in the following code. Explain in detail each line of the following code:

```
D=degree_distribution(gf)
xx=which(D>0)[-(1:10)]   # remove the first 10 prob values
lyy=log(D[xx])
lxx=log(xx)
LMI=lm(lyy~lxx)$coefficients  # line coefficients
plot(D,pch=20,cex=0.7,xlab="Degree",ylab="Frequencies",main="degrees",log="xy")
points(exp(lxx),exp(LMI[1]+LMI[2]*lxx),col="red",type="l",lwd=2)
```



- `D=degree_distribution(GRAPH)`: Se obtiene un vector D con las frecuencias relativas de cada grado,

es decir, el primer valor es la frecuencia relativa de los nodos de grado 0, el segundo valor la frecuencia de grado 1,...

- `xx=which(D>0)[-(1:10)]`: Se eliminan las frecuencias de los nodos con grado 10 o menos.
- `lyy=log(D[xx])`: Se transforma en logaritmo las frecuencias relativas de los grados restantes.
- `lxx=log(xx)`: Se transforma en logaritmo los grados.
- `LMI=lm(lyy~lxx)$coefficients`: Se obtienen los coeficientes de la regresión lineal de `lxx` sobre `lyy`.
- `plot(D,pch=20,cex=0.7,xlab="Degree",ylab="Frequencies",main="degrees",log="xy")`: Se grafican los logaritmos de las frecuencias relativas en función de los logaritmos de los grados.
- `points(exp(lxx),exp(LMI[1]+LMI[2]*lxx),col="red",type="l",lwd=2)`: Se grafica la recta de regresión.

---

d) What is the exponent of the Power Law for the degree probabilities?

```
# LO DEJO COMENTADO PORQUE AL HACER KNIT DA ERROR, LMI NO EXISTE PORQUE JAIRO HA PUESTO EL CHUNK DEL AP
#alpha = -LMI[[2]]
#alpha
# ????? EL EXPONENTE EL EL COEFICIENTE DE LA RECTA DE REGRESIÓN, PERO HAY QUE HACER OTRA VEZ LA REGRESI
```

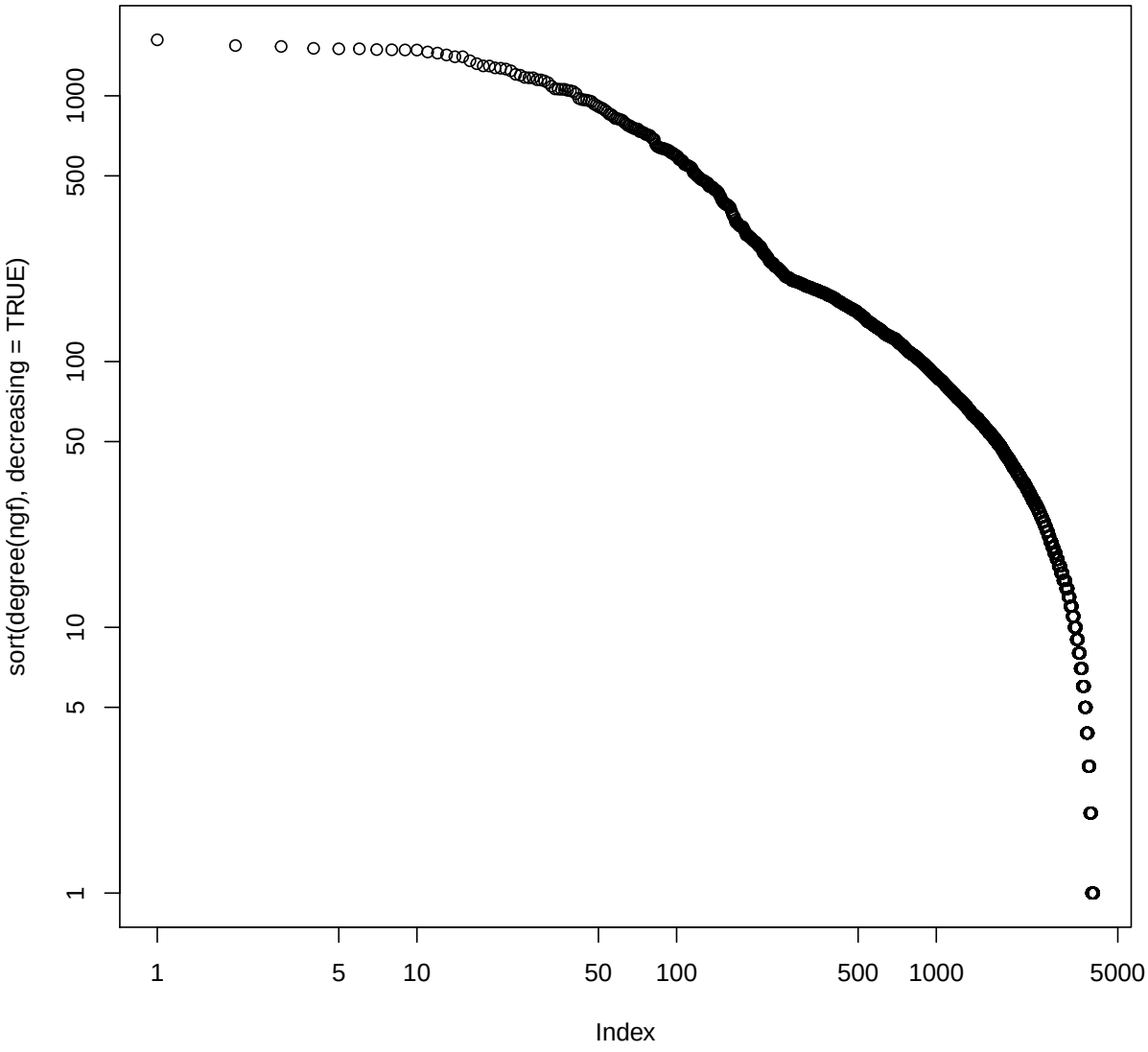
---

3) Use the routine `sample_pa()` to generate a rich-get-richer (preferential attachment) graph with similar degree distribution of the *directed* facebook graph of the file **facebook_sample_anon.txt**. Use the code similar to:

```
sample_pa(n.GRAPH, out.seq=degree(GRAPH,mode="out"))
```

Plot the degree distribution of the generated graph (log-log). What is the exponent of the power law of the generated graph for the in-degrees?

```
links=read.table("../proyecto-entrega-1/data/facebook_sample_anon.txt", header=FALSE, as.is=T)
facenet=graph_from_data_frame(d=links, directed=FALSE)
ngf <- sample_pa(vcount(facenet), out.seq=degree(facenet,mode="in"))#Lo he cambiado a "in" era "out"
plot(sort(degree(ngf), decreasing = TRUE), log = "xy")
```




```r
#Calculamos el exponente
d = degree(ngf, mode = "in")
dd = degree.distribution(ngf, mode = "in", cumulative = FALSE)
degree = 1:max(d)
probability = dd[-1]
# delete blank values
nonzero.position = which(probability != 0)
probability = probability[nonzero.position]
degree = degree[nonzero.position]
reg = lm(log(probability) ~ log(degree))
cozf = coef(reg)
alpha = -cozf[[2]]
alpha
```

```
## [1] 0.6501732
```